Context-Aware Path Ranking in Road Networks

Yang, Sean Bin; Guo, Chenjuan; Yang, Bin

Published in:
IEEE Transactions on Knowledge and Data Engineering

DOI (link to publication from Publisher):
10.1109/TKDE.2020.3025024

Publication date:
2022

Document Version
Accepted author manuscript, peer reviewed version

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2020.3025024, IEEE Transactions on Knowledge and Data Engineering

Context-Aware Path Ranking in Road Networks

Sean Bin Yang, Chenjuan Guo ✉, and Bin Yang

Abstract—Ranking paths becomes an increasingly important functionality in many transportation services, where multiple paths connecting a source-destination pair are offered to drivers. We study ranking such paths under specific contexts, e.g., at a departure time and for a specific driver. More specifically, we model ranking as a regression problem where we assign a ranking score to each path with the help of historical trajectories. The intuition is that if a driver's trajectory used path P at time t, we consider this as an evidence that path P is preferred by the driver at time t, thus should have a higher ranking score than other paths connecting the same source and destination. To solve the regression problem, we first propose an effective training data enriching method to obtain a *compact* and *diversified* set of training paths using historical trajectories, which provides a data foundation for *efficient* and *effective* learning. Next, we propose a multi-task learning framework that considers features representing both candidate paths and contexts. Specifically, a road network embedding is proposed to embed paths into feature vectors by considering both road network topology and spatial properties, such as distances and travel times. By modeling different departure times as a temporal graph, graph embedding is used to embed departure times into feature vectors. The objective function not only considers the discrepancies on ranking scores but also the reconstruction errors of the spatial properties of the paths, which in turn improves the final ranking estimation. Empirical studies on a substantial trajectory data set offer insight into the designed properties of the proposed framework, indicating that it is effective and practical in real world settings.

Index Terms—Path ranking, diversified paths, multi-task learning, road network embedding, graph embedding.

1 Introduction

Vehicular transportation reflects the pulse of a city. It plays an essential role in people's daily lives and many businesses as well as society as a whole [1]. With recent deployment of sensing technologies and continued digitization, large amounts of vehicle trajectory data are collected, which provide a solid data foundation to improve the quality of a wide variety of transportation services, such as vehicle routing [2], traffic prediction [3], and urban planning [4].

A fundamental functionality in vehicular transportation is routing. Given a source and a destination, classic routing algorithms, e.g., Dijkstra's algorithm, identify *a single optimal path* connecting the source and the destination, where the optimal path is the path with the least travel cost, e.g., the shortest path or the fastest path. However, a routing service quality study [5] shows that local drivers often choose paths that are neither shortest nor fastest, rendering classic routing algorithms often impractical in many real world routing scenarios. To contend with this challenge, a wide variety of advanced routing algorithms, e.g., skyline routing [6] and k-shortest path routing [7], are proposed to identify *a set of optimal paths*, where the optimality is defined based on, e.g., pareto optimality or top-k least costs, which provide drivers with multiple candidate paths to choose. Commercial navigation systems, such as Google Maps and TomTom, often follow a similar strategy that suggests multiple candidate paths to drivers.

Under this context, ranking such candidate paths is essential for ensuring high routing quality. Existing solutions often rely on simple heuristics, e.g., ranking paths w.r.t. their travel times. However, travel times may not always be the most important factor when drivers choose paths, and a routing quality study shows that drivers often do not choose the fastest paths [5]. In addition, existing solutions often provide the same ranking to all drivers but ignore distinct preferences which different drivers may have.

In this paper, we propose a data-driven, context-aware ranking framework *PathRank* to rank paths in road networks. More specifically, *PathRank* models ranking candidate paths as a *regression* problem—for each candidate path, *PathRank* estimates a ranking score using local drivers' trajectories, which in turn enables ranking the candidate paths w.r.t. their ranking scores. The framework is flexible where different contextual information can be accommodated. For example, when accommodating driver information, it enables personalized ranking. To enable *PathRank*, two challenges must be addressed.

Enriching Training Data: To train any regression model, we need to prepare training data. We borrow the idea often used in ranking products in online shops. If a user clicks a specific product on a webpage, it provides evidence that the user is interested in the product than other products on the same webpage. Then, the clicked vs. not clicked products are considered as positive vs. negative training data to enable traning. Similarly, if a driver's trajectory used path P from source s to destination d at time t, it is an evidence that the driver considered path P as the preferred path over other paths from s to d at time t. Thus, path P is a positive training data and should have the largest ranking score. However, trajectories only provide positive training data and we still lack negative training data. Since there often exist a large amount of paths from a source to a destination, it is thus prohibitive to include all paths other than P as the negative paths. In contrast, randomly selecting a small subset of such paths may adversely affect the training effectiveness. Thus, it is challenging to select a *compact* and *diversified* training path set to represent the negative training data. A compact set ensures training *efficiency* and a diversified set ensure training *effectiveness*.

Effective Feature Representations: Effective regression models often rely on meaningful feature representations of input data. In our setting, an input to *PathRank* is a path that is a sequence of

• *S. Yang, C. Guo and B. Yang are with the Department of Computer Science, Aalborg University, Denmark.*
E-mail: {seany, cguo, byang}@cs.aau.dk

vertices in a road network graph. Here, a meaningful feature space should take into account both the topology of the underlying road network and the spatial properties of the road network, such as distances and travel times, which may influence drivers' choices. However, no existing methods are able to capture both topological and spatial properties. In addition, it is also important to embed context information, such as departure times, into meaningful representations, where, for example, temporal closeness can be preserved. This calls far new feature learning methods.

To contend with the first challenge, we propose an effective method to generate a compact and diversified training path set. We consider different travel costs that drivers may consider, e.g., distance, travel time, and fuel consumption. Next, for each travel cost, we identify a set of *diversified*, *top-k least-cost paths*. Here, two paths are diversified if the path similarity between them is smaller than a threshold, where a number of different path similarity functions can be applied [8]. As an example, diversified top-3 shortest paths consist of three paths where the path similarity of every pair of paths is smaller than the threshold and there does not exist another set of three paths which are mutually diversified and whose total distance is shorter. Considering diversity avoids including top-3 shortest paths where they only differ slightly, e.g., only one or two edges. This method makes sure that the candidate path set is diversified because the set (i) considers multiple travel costs that a driver may consider when making routing decisions; and (ii) includes paths that are dissimilar with each other. These together represent a large feature space of the underlying road network. In addition, the set is also compact since it only includes a small number of top-k paths.

Next, we address the second challenge by proposing an end-to-end learning framework to learn feature representations of paths, which capture both topological and spatial properties. Recall that the input is a path that is represented as a sequence of vertices in a road network graph. To capture the topology of the road network, we utilize unsupervised graph embedding [9] to transform vertices into feature vectors by considering road network topology. Since recurrent neural networks (RNNs) are good at modeling sequential information and since a path is a sequence of vertices, we employ an RNN to model the sequence of the feature vectors of the vertices in a path. So far, the framework already considers the topology of the underlying road network, but still lacks spatial properties, which are not captured by classic graph embedding. To accommodate the spatial properties, we let the RNN estimate multiple values, including a ranking score of the input path and also the input path's spatial properties, such as the length, the travel time, and the fuel consumption of the path. This makes the framework a *multi-task* learning framework where the *main task* is to estimate the ranking score, which is used for the final ranking, and the *auxiliary tasks* enforce to update the feature vectors of the vertices to also capture the spatial properties of the underlying road network, which eventually also help improve the accuracy of the main task.

The proposed learning framework is flexible where contextual information can be seamlessly integrated. For example, we propose a temporal graph to model peak vs offpeak periods in different days and then departure times can be converted into feature vectors that reflect temporal closeness. We show how the temporal features can be integrated into the learning framework and thus enable temporal ranking. Similarly, when incorporating feature vectors representing drivers, we enable personalized ranking.

This paper presents the first data-driven, end-to-end solution to context-aware ranking for paths in road networks. Specifically, we make four contributions. First, we propose a method to generate a compact and diversified set of training paths which enables effective and efficient learning. Second, we propose a multi-task learning framework to enable spatial network embedding that captures not only topological information but also spatial properties. Third, we integrate contextual information embedding into the framework to enable context-aware ranking. Fourth, we conduct extensive experiments using a large real world trajectory set to offer insight into the design properties of the proposed framework and to demonstrate that the framework is effective. A preliminary four-page report on the study appeared elsewhere [10].

Paper Outline: Section 2 covers related work. Section 3 covers preliminaries. Section 4 discusses enriching training data. Section 5 proposes *PathRank*. Section 6 reports on empirical evaluations. Section 7 concludes.

2 Related Work

We review related studies on (1) learning to rank in the context of information retrieval, (2) graph representation learning, (3) machine learning techniques for path recommendation, and (4) top-k path finding.

2.1 Learning to rank

Learning to rank plays an important role in ranking in the context of information retrieval (IR), where the primary goal is to learn how to rank documents or web pages w.r.t. queries, which are all represented as feature vectors. Learning to rank methods in IR can be categorized into point-wise, pair-wise, and list-wise methods. Point-wise methods estimate a ranking score for each individual document. Then, the documents can be ranked based on the ranking scores [11]. Pair-wise methods focus on, for a given pair of documents, making a binary decision on which document is better, i.e., a relative order. Here, although we do not know the ranking scores for individual documents, we are still able to rank documents based on the estimated relative orders [12]. List-wise methods take into account a set of documents and estimate the ranking for the documents [13].

Although learning to rank techniques have been applied widely and successfully in IR, they only consider textual documents and queries and cannot be applied for ranking paths in road networks, since both graph topology and spatial properties, which are the two most important factors in road networks, are ignored. We follow the idea of the point-wise learning to rank techniques in IR and propose *PathRank* to rank paths in road networks while considering both graph topology and spatial properties.

2.2 Graph Representation Learning

Graph representation learning, a.k.a., graph embedding, aims to learn low-dimensional feature vectors for vertices while preserving graph topology structure such that the vertices with similar feature vectors share similar structural properties [9], [14], [15], [16], [17]. We distinguish two categories of methods: random walk based methods and deep learning based methods.

A representative method in the first category is DeepWalk [14]. DeepWalk first samples sequences of vertices based on truncated random walks, where the sampled vertex sequences capture the connections between vertices in the graph. Then, skip-gram model [18] is used to learn low-dimensional feature vectors based

on the sampled vertex sequences. Node2vec [9] considers higher order proximity between vertices by maximizing the probability of occurrences of subsequent vertices in fixed length random walks. A key difference from DeepWalk is that node2vec employs biased-random walks that provide a trade-off between breadth-first and depth-first searches, and hence achieves higher quality and more informative embedding than DeepWalk does.

To overcome the weaknesses of random walk based methods, e.g., the difficulty in determining the random walk length and the number of random walks, deep learning based methods utilize the random surfing model to capture contextual relatedness between each pair of vertices and preserves them into low-dimensional feature vectors for vertices [16]. Deep learning based methods are also able to take into account complex non-linear relations. GraphGAN [17] is proposed to learn vertex representations by modeling the connectivity behavior through an adversarial learning framework using a minimax game. LINE [15] does not fall into the above two categories. Instead of exploiting random walks to capture network structures, LINE [15] proposes a model with a carefully designed objective function that preserves both the first-order and second-order proximities.

However, all existing graph embedding methods consider non-spatial networks such as social networks, citation networks, and biology networks. They ignore spatial properties, e.g., distances and travel times, which are crucial features in spatial networks such as road networks. In this paper, we propose a multi-task learning framework to extend existing graph embedding to incorporate important spatial properties. Experimental results show that the graph embedding that considers spatial-properties gives the best performance when ranking paths in road networks.

2.3 Machine Learning on Spatio-Temporal Data

Machine learning has been applied to spatio-temporal data such as trajectories to improve path recommendation [2], [6], [19], [20], [21]. Personalized routing [21] and context-aware routing [2], [20] aim to identify a single, optimal path for a specific driver or under a specific context. Although such studies do not provide ranking functions directly, we derive a personalized ranking approach from [21] and compare with *PathRank* in Section 6.4. Skyline routing returns a set of non-dominated paths, which are considered to be incomparable to each other and thus no ranking is provided [6], [22]. Additional attempts have been made for estimating accurate travel time or fuel consumption distributions [3], [23], [24], [24], [25], [26], [27], which are also different from ranking paths. RoadRank [28] computes influence scores for all road segments, i.e., edges, in a road network and then ranks the edges according to the influence scores. In contrast, our paper proposes *PathRank* to rank paths, not edges. Multitask learning is applied to model different drivers' driving behavior [29] such that trajectories from a same driver can be clustered together. However, it cannot be used directly for ranking paths. In addition, one paper also considers trajectory clustering [30], which is an unsupervised learning problem. It cannot be used for solving the path ranking problem, which is a supervised learning problem.

Some traffic time series prediction methods also consider graph operations, e.g., graph convolution and graph attention, in RNNs [31], [32], [33], [34], [35], but the problem settings are different and their solutions cannot be used for ranking paths. In such models, the input to an RNN unit is a whole road network graph and the RNN units capture temporal dependency. In contrast, the input to our RNN unit is a vertex in a road network graph and the RNN units capture the spatial dependency along a path.

2.4 Top-k Queries on Road Networks

A wide variety of top-k queries on road networks exist [8], [36], [37], [38]. Top-k path selection algorithms often use simple ranking functions to rank paths [7], [8]. For example, top-k fastest path finding algorithms rank paths according to the paths' travel times. In the experiments, we compare *PathRank* with such baseline ranking functions used in top-k path finding algorithms. Some other top-k algorithms consider different problem settings. For example, top-k optimal sequenced paths aim at finding the top-k shortest paths that visit a set of points of interest (POIs) such as a post office, a bank, and a grocery store [38]. Another study considers ranking a set of POIs in a road network [37], which cannot be used for ranking paths. Probabilistic top-k shortest path queries [36], [39] rank paths w.r.t. the probability of arriving within a time budget, which is provided by end users. Our problem dose not require end users to provide such time budgets.

3 Preliminaries

3.1 Basic Concepts

A *road network* is modeled as a weighted, directed graph $G = (\mathbb{V}, \mathbb{E}, D, T, F)$. Vertex set $\mathbb{V}$ represents road intersections and road ends; edge set $\mathbb{E} \subset \mathbb{V} \times \mathbb{V}$ represents road segments. Functions D, T, and F maintain the travel costs of the edges in graph G. Specifically, function $D : \mathbb{E} \rightarrow \mathbb{R}^+$ maps each edge to its length. Functions T and F have similar signatures and maps edges to their travel times and fuel consumption, respectively.

A *path* $P = (v_1, v_2, v_3, \ldots, v_X)$ is a sequence of X vertices where $X > 1$ and each two adjacent vertices must be connected by an edge in $\mathbb{E}$. We use $P.s$ and $P.d$ to denote the source and the destination of path P.

A *trajectory* $\boldsymbol{T} = (p_1, p_2, p_3, \ldots, p_Y)$ is a sequence of GPS records pertaining to a trip, where each GPS record $p_i = (location, time)$ represents the location of a vehicle at a particular timestamp. The GPS records are ordered according to their corresponding timestamps, where $p_i.time < p_j.time$ if $1 \leq i < j \leq Y$.

Map matching [40] is able to map a GPS record to a specific location on an edge in the underlying road network, thus aligning a trajectory $\boldsymbol{T}$ with a path in the underlying road network, denoted as $\boldsymbol{T}.P$. We call such paths *trajectory paths*. In addition, a trajectory $\boldsymbol{T}$ is also associated with a driver identifier, denoted as $\boldsymbol{T.driver}$, indicating who made the trajectory. From trajectory $\boldsymbol{T}$, we know that driver $\boldsymbol{T.driver}$ used path $\boldsymbol{T}.P$ at time $\boldsymbol{T}.p_1.time$. Thus, path $\boldsymbol{T}.P$ is considered as a ground truth path under the contexts, i.e., for driver $\boldsymbol{T.driver}$ at time $\boldsymbol{T}.p_1.time$.

Path Similarities: Multiple similarity functions [2], [8], [21], [41] are available to calculate the similarity between two paths, where the most popular functions belong to the Jaccard similarity function family, in particular, the weighted Jaccard similarity [2], [21]. In this paper, we use the weighted Jaccard Similarity (see Equation 1) to evaluate the similarity between two paths.

$$sim\,(P_1, P_2) = \frac{\sum_{e \in P_1 \cap P_2} G.D(e)}{\sum_{e \in P_1 \cup P_2} G.D(e)} \tag{1}$$

Here, we use $P_1 \cap P_2$ and $P_1 \cup P_2$ to represent two edge sets: edge set $P_1 \cap P_2$ consists of the edges that appear in both P_1 and P_2;

and edge set $P_1 \cup P_2$ consists of the edges that appear in either P_1 or P_2. Recall that function $G.D(e)$ returns the length of edge e. Then, the intuition of the weighted Jaccard similarity is two-fold: first, the more edges the two paths share, the more similar the two paths are; second, the longer the shared edges are, the more similar the two paths are. Note that the proposed *PathRank* is a generic path ranking framework, which is able to easily incorporate other similarity functions.

Ranking scores: Given a trajectory path P and another path P' that also connects $P.s$ and $P.d$, we use the similarity between the two paths $sim(P, P')$ to represent the ranking score of P'. Since we consider trajectory paths as the ground truth path under the contexts, the more similar P' is w.r.t. P, the higher similarity score P' should have and thus should rank higher. The trajectory path P itself always has a ranking score of 1 and thus ranks the highest among all paths connecting $P.s$ and $P.d$.

3.2 Problem Definition

Given a set of N candidate paths $\mathbb{P}$ that connect the same source and destination and optional contexts such as a departure time and a driver identifier, we aim at (1) estimating a ranking score $sim(P, P'_i)$ for each candidate path $P'_i \in \mathbb{P}$; and (2) providing a ranked list of the candidate paths $\langle P'_1, P'_2, \ldots, P'_N \rangle$, such that $sim(P, P'_i) \geq sim(P, P'_j)$ when $1 \leq i < j \leq N$.

3.3 *PathRank* Overview

Fig. 1 shows an overview of the proposed *PathRank*. We distinguish a training phase and a testing phase. The training phase employ historical trajectories to train *PathRank*, and we use the trained *PathRank* in the testing phase.

We proceed to elaborate the training phase. Given a set of historical trajectory, we first map match them to obtain their corresponding *trajectory paths*. The trajectory paths are fed into the *Training Data Enrichment* module where an enriched training data set is generated. Specifically, for each trajectory path P, the training data enrichment module generates a compact and diversified set $\mathcal{PS}$ of candidate paths such that each candidate path $P' \in \mathcal{PS}$ also connects the same source and destination of the trajectory path P. In addition, for each path $P' \in \mathcal{PS}$, the module computes a similarity score $sim(P, P')$ as the ground truth ranking score of P'. Thus, the output of the training data enrichment module is a set of "candidate path" and "ranking score" pairs, denoted as $\{P', sim(P, P')\}$, where the ranking scores are labels. This set is used as the input for the *PathRank*.

In the training phase, for each training instance $(P', sim(P, P'))$, the *Spatial Network Embedding Module* embeds each vertex in candidate path P' into a feature vector. This transfers path P' into a sequence of feature vectors, which is then fed into a *Recurrent Neural Network* (RNN). In addition, *Context Embedding Module* embeds additional contextual information such as departure time and driver identifiers into feature vectors, which is also fed into the RNN. Then, the RNN estimates a ranking score of P'. An objective function is designed to measure the discrepancy between the estimated ranking score and the ground truth ranking score $sim(P, P')$.

In the testing phase, we use the trained *PathRank* to rank candidate paths. Given a source, a destination, and optional contexts, advanced routing algorithms or commercial navigation systems are able to provide multiple candidate paths, which are used as candidate paths. Next, *PathRank* takes as input each candidate path and returns an estimated ranking score. Finally, we rank the candidate paths according to their estimated ranking scores.

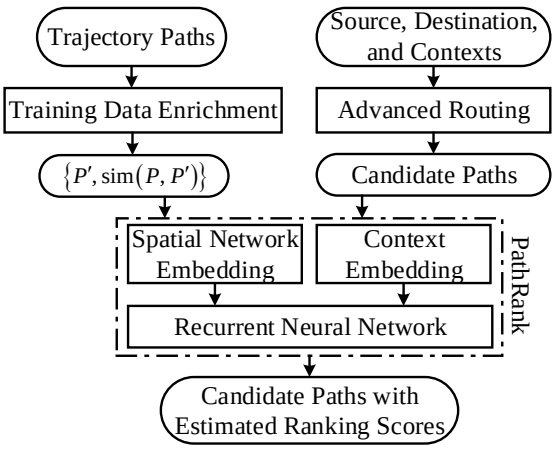


Fig. 1: Solution Overview.

4 Training Data Enrichment

We proceed to elaborate how to generate a compact and diversified set of training paths for a trajectory path.

4.1 Intuitions

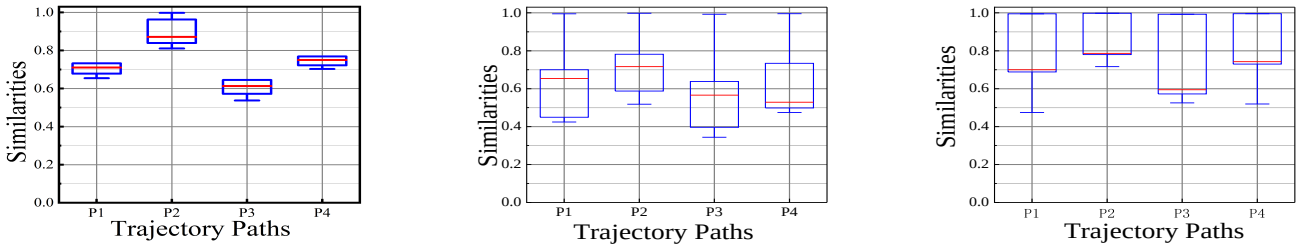
Ranking paths is similar to ranking products in online shops. If a user clicks a specific product, it provides evidence that the user is interested in the product than other similar products. Similarly, a trajectory path P from a source s to destination d at time t also provides strong evidence that a driver prefers path P than other paths that connect s to d at time t. The main difference is that, in online shops, the other similar products, i.e., competitor products, can be obtained explicitly, e.g., those products that are shown to the user in the same web page but are not clicked by the user. Based on the positive and negative training data, i.e., the products that are clicked and not clicked by the user, effective learning mechanism, e.g., learning to rank [42], [43], is available to learn an appropriate ranking function. However, in our setting, other candidate paths are often unknown and implicit because we do not know when the driver made the decision to take path P, what other paths were in driver's mind. Thus, the main target of the training data enrichment module is to generate a set of paths $\mathcal{PS}$ that include the other paths that the driver has considered. We call $\mathcal{PS}$ *competitive path set*.

A naive way to generate the competitive path set is to simply include all paths from s to d. This is infeasible to use in real world settings since the competitive path set may contain a huge number of paths in a city-level road network graph, which in turn makes the training prohibitively inefficient. Thus, we aim to identify a *compact* competitive path set, where only a small number of paths, e.g., less than 10 paths, are included. However, we cannot just randomly choose a small number of paths. We need to carefully choose such paths to resemble "the unclicked products" in online shopping.

4.2 Top-k Shortest Paths

The first strategy is to employ a classic top-k shortest path algorithm, e.g., Yen's algorithm [7], to include the top-k shortest paths from s to d into the competitive path set $\mathcal{PS}$. This provides us a

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2020.3025024, IEEE Transactions on Knowledge and Data Engineering



(a) Top-9 Shortest Paths (b) Diversified Top-9 Shortest Paths (c) Multi-cost Diversified Paths

Fig. 2: Similarity Spreads of Different Strategies.

compact set. In addition, this strategy is simple and efficient since a wide variety of efficient algorithms are available to generate top-k shortest paths in the literature [7], [44]. However, a serious issue of this strategy is that the top-k shortest paths are often highly similar. For example, the top-2 shortest paths may only differ with one or two edges. Thus, their similarities w.r.t. the ground truth, trajectory path P, are also similar, which adversely affect the effectiveness of the subsequent ranking score regression.

For example, we randomly choose four trajectory paths with different sources and destinations. For each trajectory path, we identify its origin and destination. Then, we use the origin and destination to generate top-9 shortest paths. Then, we compute the competitive paths' similarities w.r.t. the trajectory path. Figure 2a shows the box plots of the similarities per trajectory path. We observe that the similarities often only spread over a very small range. For example, for the first trajectory path P_1, its corresponding top-9 shortest paths have similarities spreading from 0.65 to 0.75.

If the similarities of competitive paths only spread over a small range, they only provide training instances for estimating ranking scores in the small range, which may make the trained model unable to make accurate estimations for ranking scores outside the small range. Thus, an ideal strategy should be providing a set of training paths whose similarities cover a large range. In other words, we aim at getting a diversified competitive path set to ensure effectiveness. To this end, we propose the second strategy using the diversified top-k shortest paths [8].

4.3 Diversified Top-k Shortest Paths

Diversified top-k shortest paths finding aims at identifying top-k shortest paths such that the paths are mutually dissimilar, or diverse, with each other. First, we always include the shortest path into the diversified top-k shortest path set, say $DkPS$. Next, we iteratively check the next shortest path P_i until we have included k paths in $DkPS$ or we have checked all paths connecting the source and destination. When checking the next shortest path P_i, we include P_i into $DkPS$ if the similarity between P_i and each existing path in $DkPS$ is smaller than a threshold δ. This means that P_i is sufficiently dissimilar with the paths in $DkPS$, thus making sure that $DkPS$ is a diversified top-k shortest path set. The smaller the threshold δ is, the more diverse the paths in $DkPS$ are. However, if the threshold δ is too small, it may happen that less than k diverse shortest paths or even only the shortest path can be included in $DkPS$.

Figure 2b shows the similarities of the same four trajectory paths when using diversified top-9 shortest paths with threshold $\delta = 0.8$. We observe that the similarities spread over larger ranges compared to Figure 2a when using classic top-k shortest paths.

4.4 Considering Multiple Travel Costs

Recent studies on personalized routing [2], [21], [45] suggest that a driver may consider different travel costs, e.g., travel time, distance, and fuel consumption, when making routing decisions. This motivates us to consider multiple travel costs, but not only distance, when generating competitive path sets. The first option to do so is to use Skyline routing [6], which is able to identify a set of pareto-optimal paths, a.k.a., Skyline paths, when considering multiple travel costs. However, Skyline routing also suffers the high similarity problem that the classic top-k shortest paths have—it often happens that the skyline paths are mutually similar, which may adversely affect the training effectiveness.

We propose a simple yet effective approach. We run the diversified top-k shortest paths x times where each time we consider a specific travel cost. Then, we use the union of the diverse paths as the final competitive path set PS. For example, when considering three travel costs, i.e., distances, travel times, and fuel consumption, we set $x = 3$ and identify the diversified top-k shortest, fastest, and most fuel efficient paths, respectively. Then, the union of the diversified top-k shortest, fastest, and most fuel efficient paths is used as the final competitive path set PS.

Since we run the diversified top-k shortest path finding multiple times for different travel costs, we can use a small k for each run. For example, when we set $k = 3$ and consider three travel costs, this makes PS also consist of up to 9 paths including the top-3 shortest, fastest, and most fuel efficient paths. Figure 2c shows the similarities of the same four trajectory paths when using the multi-cost diversified paths that include the top-3 shortest, fastest, and most fuel efficient paths. The similarities in Figure 2c spread over larger ranges and the ranges are closer to 1. This is preferred since it helps us to distinguish the rankings of "good enough" candidate paths.

To summarize, we use multi-cost, diversified top-k least-cost paths as the compact competitive path set PS for each trajectory path P. We use paths in PS and trajectory path P together as the training data, denoted as $\{(P'_i, sim_i)\}$. Here, path $P'_i \in PS \cup \{P\}$ is associated with a ranking score label $sim_i = sim(P'_i, P)$. If P'_i is a trajectory path, its ranking score sim_i is 1, which serves as a positive training data. Otherwise, the ranking score is smaller than 1, which serves as a negative training data. After identifying competitive path sets for all trajectory paths, we use $\{(P'_i, sim_i)\}$ as the training data for *PathRank*. If we do not enrich training data and only use trajectory paths for training, then they all have ranking score of 1, making it impossible to rank different paths.

5 Ranking Framework

We propose an end-to-end deep learning framework to estimate similarity scores for paths. We first propose a basic framework that

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2020.3025024, IEEE Transactions on Knowledge and Data Engineering

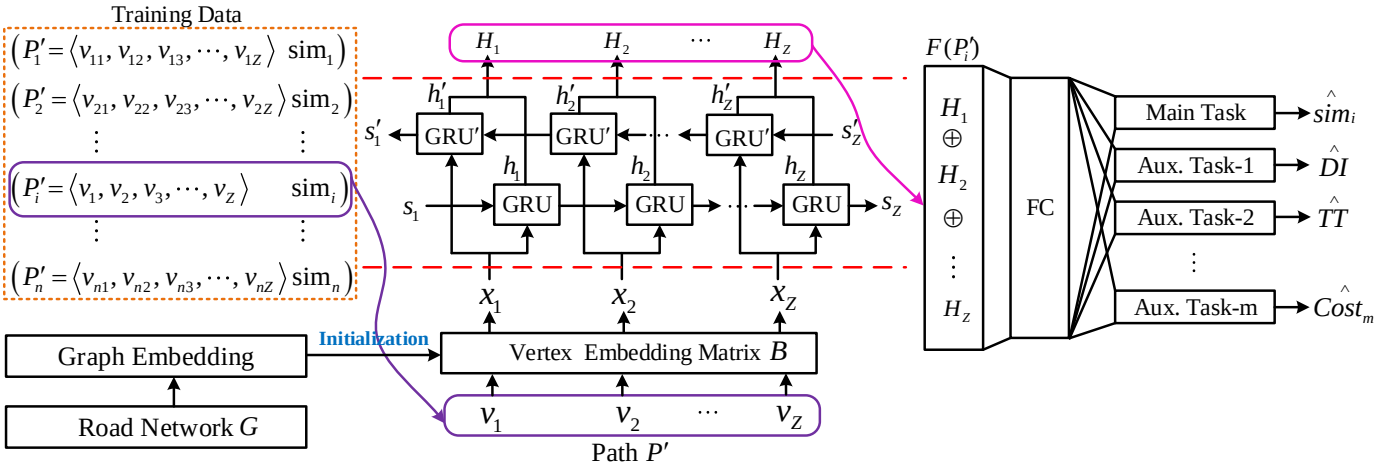


Fig. 3: Basic Framework of *PathRank*.

consists of a spatial network embedding network and a recurrent neural network. Next, we extend the basic framework with the help of contextual embedding by considering two contexts, i.e., departure time and driver identifiers.

5.1 Basic Framework

Recall that the input for *PathRank* is a path, i.e., competitive path P_i', and the label of the input is its ranking score, i.e., similarity, sim_i. To solve the ranking score regression problem, a prerequisite is to represent the input path P_i' into an appropriate feature space. To this end, we propose to use a vertex embedding network to convert each vertex in the input path to a feature vector. Since a path is a sequence of vertices, after vertex embedding, the path becomes a sequence of feature vectors. Next, since recurrent neural networks (RNNs) are capable of capturing dependency for sequential data, we employ an RNN to model the sequence of feature vectors. The RNN finally outputs an estimated ranking score, which is compared against the ground truth ranking score sim_i. This results in the basic framework of *PathRank*, which consists of two neural networks—a vertex embedding network and a recurrent neural network (RNN), as shown in Figure 3.

5.1.1 Vertex Embedding

We represent a vertex v_i in road network graph G as a one-hot vector $q_i \in \mathbb{R}^N$, where N represents the number of vertices in G, i.e., $N = |G.\mathbb{V}|$. Specifically, the i-th vertex v_i in graph G is represented as a vector q_i where the i-th bit is 1 and the other $N-1$ bits are 0. Vertex embedding employs an embedding matrix $B \in \mathbb{R}^{M \times N}$ to transfer a vertex's one-hot vector q_i into a new feature vector $x_i = Bq_i \in \mathbb{R}^M$. The feature vector is often in a smaller space, where $M < N$.

Given a competitive path $P_i' = \langle v_1, v_2, \ldots, v_Z \rangle$, we apply the same embedding matrix B to transfer each vertex to a feature vector. Thus, the competitive path P is represented as a sequence of features $\langle x_1, x_2, \ldots, x_Z \rangle$, where $x_j = Bq_j$ and $1 \leq j \leq Z$.

Next, we elaborate different means of obtaining embedding matrix B. An naive method to obtain B is to simply initialize a random matrix, which is then updated through back-propagation in the training phase. However, the naive method does not consider the graph topology and spatial properties, which hinders accuracy.

Capturing Graph Topology with Graph Embedding: Graph embedding, e.g., DeepWalk [14], node2vec [9], LINE [15], GraphGAN [17], aims at learning low-dimensional, feature vectors of vertices in a graph by taking into account the graph topology. A typical way to enable graph embedding is to mimic the way of embedding words for natural languages [9], [14]. In particular, multiple vertex sequences can be generated by using random walks, where random walks can consider edge weights or ignore edge weights. Next, vertices are considered as words and the generated vertex sequences are considered as sentences, which enables the use of word embedding techniques to generate embeddings for vertices. Since the vertex sequences are generated by applying random walks on the graph, the obtained vertex embedding actually already takes into account the graph topology. The output of graph embedding is an embedding matrix B that considers graph topology.

We propose two different strategies to incorporate graph embedding into the framework. First, we simply apply an existing graph embedding method, e.g., DeepWalk or node2vec, to obtain embedding matrix B that embeds a one-hot representation of a vertex to a low dimensional feature vector. Then, we use the feature vector as the input to the RNN. This means that *PathRank* only includes an RNN module, whose inputs are sequences of feature vectors, and the vertex embedding module only provides the inputs and are then disconnected from *PathRank*.

Second, inspired by the well-known practice of unsupervised pre-training [46], we use the embedding matrix obtained from an existing graph embedding method to initialize the embedding matrix B in the vertex embedding module in *PathRank*. This allows *PathRank* to update the embedding matrix B during training such that it not only captures the graph topology but also better fits the similarity regression.

Capturing Spatial Properties with Multi-Task Learning: Although many vertex embedding algorithms exist, they are only able to capture graph topology because they only focus on graphs representing, e.g., social networks and citation network. In other words, they do not consider graphs representing spatial networks such as road networks. However, in road network graphs, many spatial attributes, in addition to topology, are also very important. For example, distances and travel times between two vertices are crucial features for road networks and also influence drivers' path choices. To let the graph embedding also maintain the spatial properties, we design a multi-task learning framework using pre-trained graph embedding.

We first employ an existing graph embedding algorithm to initialize the vertex embedding matrix B in the vertex embedding module of *PathRank*. This pre-trained embedding matrix captures the graph topology. Next, we try to update B such that it also captures relevant spatial properties during training. To this end, we employ multi-task learning principles, where the main task is to estimate similarity and the auxiliary tasks are to reconstruct travel costs of competitive paths which help learning an appropriate embedding matrix B that also considers spatial properties of the underlying road network.

5.1.2 RNN

After vertex embedding, a path is represented by a sequence of feature vectors $\langle x_1, x_2, \ldots, x_Z \rangle$. The feature sequence represents the flow of travel on path P'_i. As a recurrent neural network (RNN) is known to be effective for modeling sequences, we fed the feature sequence $\langle x_1, x_2, \ldots, x_Z \rangle$ into an RNN. Specifically, we employ a bidirectional gated recurrent neural network (BD-GRU) [47] to capture the sequential dependencies in both the direction and the opposite direction of the travel flow on path P'_i.

We consider the direction of the travel flow first, i.e., from left to right. A GRU unit learns sequential correlations by maintaining a hidden state $h_j \in \mathbb{R}^Q$ at position j, which can be regard as an accumulated information of the positions to the left of position j. Specifically, $h_j = GRU(x_j, h_{j-1})$, where x_j is the input feature vector at position j and h_{j-1} is the hidden state at position $j-1$, i.e., the hidden state of the left position. More specifically, the GRU unit is composed of the following computations.

First, the GRU unit employs a reset gate r_j, shown in Equation 2, to decide how much information from the previous position should be forgotten. Equation 2 computes r_j, which is a value between 0 and 1, meaning that the reset gate may fully forget to fully remember. The GRU then uses a similar gate called update gate to compute z_j using Equation 3. Both the reset and update gates are contributed to control how much information from the left hidden states should be considered in order to make the final similarity score estimation accurate. More specifically, In Equation 4, the GRU computes an internal state $\tilde{h}_j$ that considers both inputs x_j and h_{j-1}. Here, the output of the reset gate r_j is used to control how much we want to consider the output from the previous position h_{j-1}. Finally, In Equation 5, the GRU uses the update gate z_j to combine the internal state $\tilde{h}_j$ and the output from the previous position h_{j-1}, which produces the output state h_j for the current GRU unit at position j. By doing this, it is possible to remember and forget left hidden states which are found to be relevant and irrelevant for the final similarity score estimation.

$$\mathbf{r}_j = \sigma\left(\mathbf{W}_r x_j + \mathbf{U}_r \mathbf{h}_{j-1}\right) \tag{2}$$

$$\mathbf{z}_j = \sigma\left(\mathbf{W}_z x_j + \mathbf{U}_z \mathbf{h}_{j-1}\right) \tag{3}$$

$$\tilde{\mathbf{h}}_j = \phi\left(\mathbf{W}_h x_j + \mathbf{U}_h\left(\mathbf{r}_j \odot \mathbf{h}_{j-1}\right)\right) \tag{4}$$

$$\mathbf{h}_j = \mathbf{z}_j \odot \mathbf{h}_j + (1 - \mathbf{z}_j) \odot \tilde{\mathbf{h}}_j \tag{5}$$

where σ is the logistic function, and $\odot$ denotes Hadamard product and ϕ is hyperbolic tangent function. x_j and $\mathbf{h}_j$ are the feature vector and hidden state at position j, respectively. $\mathbf{W}_r$, $\mathbf{W}_z$, $\mathbf{W}_h$, $\mathbf{U}_r$, $\mathbf{U}_z$ and $\mathbf{U}_h$ are parameters to be learned.

For the opposite direction of the travel flow, i.e., from right to left, we apply another GRU to generate hidden state $\mathbf{h}'_j = GRU'(x_j, \mathbf{h}'_{j+1})$. Here, the input consists of the feature vector at position j and the hidden state at position $j+1$, i.e., the right hidden state.

The final hidden state H_j at position j is the concatenation of the hidden states from both GRUs, i.e., $H_j = \mathbf{h}_j \oplus \mathbf{h}'_j$ where $\oplus$ indicates the concatenation operation. We stack all outputs from the BD-GRU units into a long feature vector $F(P'_i) = \langle H_1 \oplus H_2 \oplus \ldots \oplus H_Z \rangle$ where $\oplus$ indicates the concatenation operation. Now, the competitive path P'_i is converted to a feature vector $F(P'_i)$.

5.1.3 Fully Connected Layer

For each competitive path P'_i, we apply a fully connected layer with weight vector $W_{FC} \in \mathbb{R}^{|F(P'_i)| \times X}$ to produce a vector of X values, including the estimated similarity score $\hat{sim}_i$ and a number of spatial properties, such as travel time, distance, and fuel consumption.

5.1.4 Loss Function

To enable the multi-task learning framework, in the final fully connected layer, *PathRank* not only estimates a similarity score but also reconstruct the spatial properties of the corresponding competitive path P'_i, such as the distance, travel time, and fuel consumption of P'_i. The loss function for the multi-task learning framework is defined in Equation 6.

$$\begin{aligned}\mathcal{L}(\mathbf{W}) = &\frac{1}{|n|}[(1-\alpha) \cdot \sum_{i=1}^{n} \left(\hat{sim}_i - sim_i\right)^2 + \\ &\alpha \cdot \sum_{i=1}^{n} \sum_{k=1}^{m} \left(\hat{y}_i^{(k)} - y_i^{(k)}\right)^2] + \lambda \|\mathbf{W}\|_2^2\end{aligned} \tag{6}$$

The first term of the loss function measures the discrepancy between the estimated similarity $\hat{sim}_i$ and the ground truth similarity sim_i. We use the average of square error to measure the discrepancy, where n is the total number of competitive paths we used for training. The second term of the loss function represents auxiliary tasks that consider the discrepancies between the actual spatial properties vs. the estimated spatial properties. More specifically, $\hat{y}_i^{(k)}$ and $y_i^{(k)}$ denote the estimated cost of the k-th auxiliary task and the ground truth of the k-th auxiliary task, respectively. For example, when considering distance, travel time, and fuel consumption, we set m to 3; and $\hat{y}_i^{(k)}$ and $y_i^{(k)}$ represent the estimated and ground truth distance, travel time, or fuel consumption of the i-th competitive path P'_i. α is a hyper parameter that controls the trade-off between main task and auxiliary tasks. Finally, the loss function uses a L2 regularizer on all learnable parameters in the model, including the embedding matrix B, multiple matrices used in BD-GRU, and the matrix in the final fully connected layer W_{FC}.

5.2 Advanced Framework

Path ranking is often context dependent. For example, during peak vs off-peak hours, drivers may consider different paths as the best paths. To accommodate such contexts, we design an advanced framework to extend the basic framework with the help of contextual embedding by considering a departure time t and a driver ID k. The advanced *PathRank* framework is shown in Figure 4.

We proceed to describe the embedding of various contexts such as departure times and driver IDs. Departure time is an important context when drivers make routing decisions as it often

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2020.3025024, IEEE Transactions on Knowledge and Data Engineering

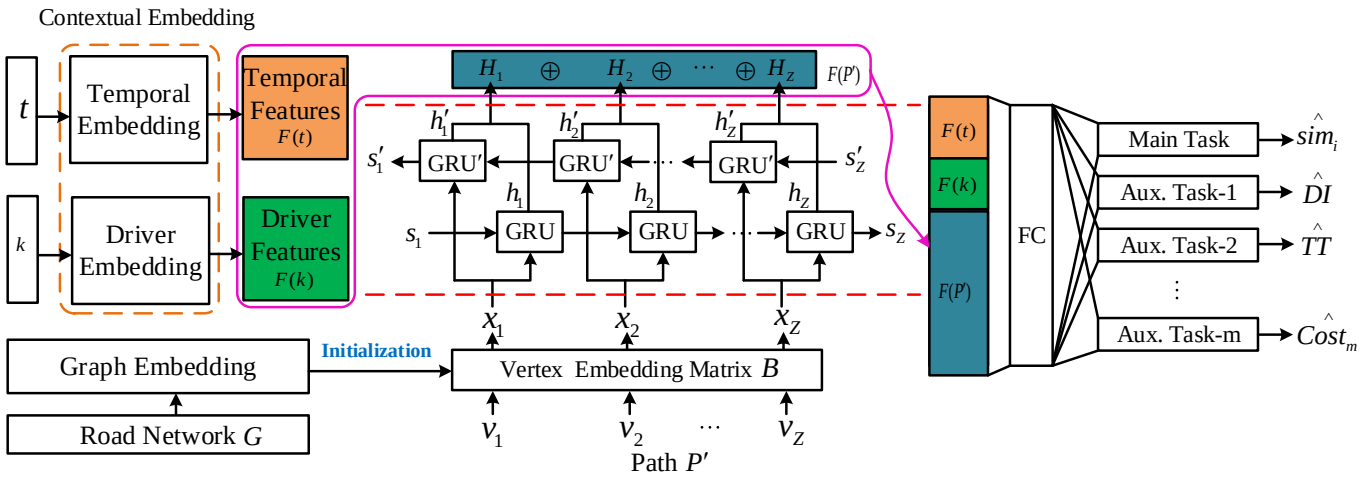


Fig. 4: Advanced *PathRank* Overview.

correlates with traffic conditions which affect heavily drivers' routing decisions. We aim at embedding departure time into a meaningful feature space such that the ranking model is able to take into account departure time.

We first partition a day into five intervals—a morning peak interval, an afternoon peak interval, and three off-peak intervals. Various methods are available to partition a day into peak and off-peak intervals [6], [48]. For example, an example partition is shown at the top of Figure 5.

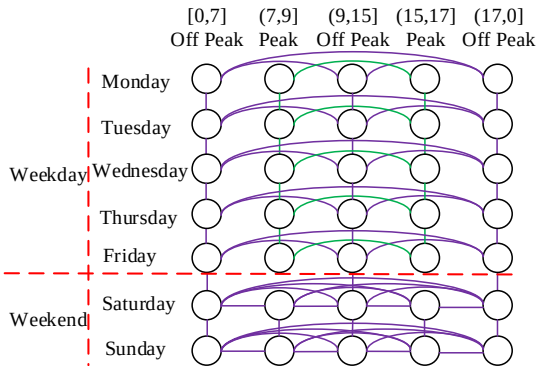


Fig. 5: Temporal Graph.

Next, we construct a temporal graph based on the intervals in different days in a week, as shown in Figure 5. In the graph, each node represents an interval in a specific day. If two nodes are supposed to have similar traffic, we connect the two nodes by an edge. For each weekday, we assume that the peak intervals have similar traffic situations, thus we connect the two nodes representing the two peak intervals. Similarly, we connect the nodes representing offpeak hours in the same weekday. We also connect the two nodes which represent the same interval but on two adjacent weekdays, e.g., two nodes representing the morning peak on Wednesday and the morning peak on Thursday. For each weekend, we connect all nodes in the weekend. Between the two weekends, we connect the two nodes representing the same interval.

Based on the temporal graph, we apply graph embedding to embed the nodes in the temporal graph into feature vectors. Given a departure time t, we first identify the node $node(t)$ that t belongs to in the temporal graph and then obtain its embedding $F(t) = GraphEmbed(node(t))$. Like the road network embedding, we allow the learning framework to update the temporal graph embedding to better fit the ranking score regression.

We use one-hot encoding to convert driver ID k into a multidimensional feature vector $F(k)$.

To incorporate the context features into the framework, we concatenate the context features with the path feature. Specifically, assume that the competitive path P'_i corresponds to a trajectory path that is made by driver k at departure time t, the final feature vector for the competitive path P'_i is $F(t) \oplus F(k) \oplus F(P'_i)$. Then, similar to the basic framework, the feature vector is fed into a fully connected layer to estimate similarity scores and different spatial properties using the same loss function shown in Equation 6.

6 Experiments

We conduct a comprehensive empirical study to investigate the effectiveness of the proposed *PathRank* framework.

6.1 Experiments Setup

6.1.1 Road Network and Trajectories

We obtain the Danish road network from OpenStreetMap, which consists of 667,950 vertices and 818,020 edges. We use a substantial GPS data set occurred on the road network, which consists of 180 million GPS records for a two-year period from 166 drivers. The sampling rate of the GPS data is 1 Hz (i.e., one GPS record per second) and each GPS record is associated with a driver identifier. We split the GPS records into 22,612 trajectories representing different trips. A well-known map matching method [40] is used to map match the GPS trajectories such that for each trajectory, we obtain its corresponding trajectory path.

6.1.2 Travel costs

We consider three travel costs: travel distance (DI), travel time (TT), and fuel consumption (FC). The travel distances are computed based on the geometric information provided by OpenStreetMap. Travel times are obtained as the difference between the times of the last and first GPS records of the trajectories. We use the SIDRA-running model to compute fuel consumption based on the speeds that are obtained from the available GPS records [49]. A recent benchmark indicates that the SIDRA-running is appropriate for this purpose [50].

6.1.3 Ground Truth Data

We split the trajectories into three sets—70% for training, 10% for validation, and 20% for testing. The distributions of the cardinalities of the trajectory paths in training, validation, and testing sets are shown in Figure 6.

For each trajectory T, we obtain its source s, destination d, and the trajectory path P_T. Then, we employ seven different strategies to generate seven sets of competitive paths according to the source-destination pairs (s, d).

1) Top-k shortest paths (*TkDI*);
2) Top-k fastest paths (*TkTT*);
3) Top-k most fuel efficient paths (*TkFC*);
4) Diversified top-k shortest paths (*D-TkDI*);
5) Diversified top-k fastest paths (*D-TkTT*);
6) Diversified top-k most fuel efficient paths (*D-TkFC*);
7) Diversified, multi-cost top-k paths (*D-TkM*).

For each competitive path P, we employ weighted Jaccard similarity $sim(P, P_T)$ as the *ground truth* ranking score of path P.

When training and validation, we use the competitive path set generated by a specific training data generation strategy to train a *PathRank* model. Thus, we are able to train seven different *PathRank* models using the same set of training and validation trajectories, but seven different sets of competitive paths.

When testing, to make the comparison among different *PathRank* models fair, for each testing trajectory, we merge all competitive paths generated by the 7 different strategies and randomly choose 10 paths from them. This makes sure that (1) *PathRank* models that are trained on different training data sets are tested against on the same set of competitive paths; (2) a *PathRank* model that is trained on a specific strategy is tested against competitive paths that are not generated from the same strategy.

6.1.4 PathRank *Frameworks*

We consider different variations of *PathRank*.

1) *PR-B*: the vertex embedding just employs a random initialized embedding matrix B, which ignores the graph topology. We also let $\alpha = 0$, meaning that *PR-B* has a single task on estimate similarity scores, where α is a parameter that controls the relative importance between the main task and auxiliary tasks as shown in Equation 6.
2) *PR-A1*: vertex embedding employs graph embedding that considers graph topology, but the vertex embedding is static and is not updated during training. Only the main task is considered, i.e., $\alpha = 0$.
3) *PR-A2*: similar to *PR-A1*, graph embedding is used. In addition, the vertex embedding is updated during training. Only the main task is considered, i.e., $\alpha = 0$.
4) *PR-A2-Mx*: Similar to *PR-A2*, graph embedding is used and the vertex embedding is updated during training. In addition, multi-task learning that considers spatial properties is used. We use *PR-A2-M*x to indicate a *PathRank* model that uses an objective function considering x spatial properties, i.e., x auxiliary tasks.
5) *PRC*: the advanced framework *PRC* with contextual embedding and multi-task learning.

For all frameworks that use graph embedding, i.e., *PR-A1*, *PR-A2*, *PR-A2-M*x and *PRC*, we choose node2vec [9] as the graph embedding method. Node2vec is a generic random walk based graph embedding method, which outperforms alternative methods such as DeepWalk [14] and LINE [15]. When new, better unsupervised graph embedding method becomes available, it can be easily integrated into *PathRank* to replace node2vec.

6.1.5 Parameters

When generating diversified top-k paths, we consider two different similarity thresholds δ—0.6 and 0.8. A smaller threshold enforces more diversified paths. However, it is also more likely that we cannot identify k paths that are significantly diversified paths, especially when k is large. Recall that the vertex embedding utilizes a embedding matrix $B \in \mathbb{R}^{M \times N}$ to embed each vertex into a M-dimensional feature vector, where N is the number of vertices. We consider two settings of M, namely 64 and 128. For the multi-task learning framework, we vary α from 0, 0.2, 0.4, 0.6, to 0.8 to study the effect on learning additional spatial properties.

We summary different parameter settings in Table 1, where the default values are shown in bold.

TABLE 1: Parameters of *PathRank*

Parameters	Values
Similarity Threshold δ	0.6, **0.8**
Vertex Embedding Feature Size M	64, **128**
Multi-task Learning Parameter α	0, 0.2, 0.4, 0.6, 0.8

6.1.6 Evaluation Metrics

We evaluate the accuracy of the proposed *PathRank* framework based on two categories of metrics. The first category includes metrics that measure how accurate the estimated ranking scores w.r.t. the ground truth ranking scores. This category includes Mean Absolute Error (MAE) and Mean Absolute Relative Error (MARE). Smaller MAE and MARE values indicate higher accuracy. Specifically, we have

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|x_i - \hat{x}_i|; \quad \text{MARE} = \frac{\sum_{i=1}^{n}|x_i - \hat{x}_i|}{\sum_{i=1}^{n}|x_i|} \tag{7}$$

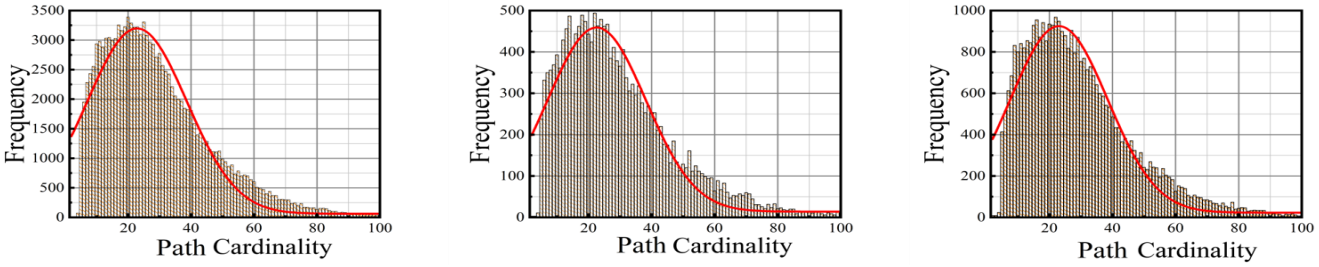
where x_i and $\hat{x}_i$ represent the ground truth ranking score and the estimated ranking score, respectively; and n is the total number of estimations.

The second category includes Kendall rank correlation coefficient (denoted by τ) and Spearman's rank correlation coefficient (denoted by ρ), which measure the similarity, or consistency, between a ranking based on the estimated ranking scores and a ranking based on the ground truth ranking scores. Sometimes, although the estimated ranking scores deviate from the ground truth ranking scores, the two rankings derived by both scores can be consistent. In this case, we consider the estimated ranking scores also accurate, since we eventually care the final rankings of the candidate paths but not the specific ranking scores for individual candidate paths. Both τ and ρ are able to measure how consistent between the two rankings. The higher the values are, the more consistent the two rankings are. If the two rankings are identical, both τ and ρ values are 1. Specifically, we have

$$\tau = \frac{N_{con} - N_{dis}}{n(n-1)/2}; \quad \rho = 1 - \frac{6\sum_{i=1}^{n} d_i^2}{n(n^2-1)} \tag{8}$$

Assume that we have a set of $n = 3$ candidate paths $\{P_1, P_2, P_3\}$, the ground truth ranking is $\langle P_1, P_2, P_3\rangle$, and the estiamted ranking is $\langle P_2, P_3, P_1\rangle$.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2020.3025024, IEEE Transactions on Knowledge and Data Engineering



(a) Training Set (b) Validation Set (c) Testing Set

Fig. 6: Cardinalities of the trajectory paths.

In τ, N_{con} and N_{dis} represent the number of path pairs are consistent and inconsistent in the two rankings. We have $N_{con} = 1$ since in both ranking, P_2 appears before P_3; and $N_{dis} = 2$ since P_1 appears before P_3 in the ground truth ranking, but P_3 appears before P_1 in the estimated ranking. Similarly, the orderings between P_1 and P_2 are also inconsistent in two rankings.

In ρ, d_i represents the rank difference on the i-th competitive path in both rankings. Following the running example, we have $d_1 = 1 - 3 = -2$ because path P_1 has rank 1 and rank 3 in both rankings, respectively.

6.1.7 Baselines

Baseline Ranking Heuristics: We consider seven baseline ranking heuristics. The first three baseline ranking heuristics consider a single cost, i.e., ranking the candidate paths according to only distances (DI), travel times (TT), and fuel consumption (FC). The three baseline ranking heuristics represent the ranking part of top-k path selection algorithms. For example, DI represents the ranking part of top-k shortest path selection.

Next, we consider four more baseline ranking heuristics that consider multiple travel costs—ranking the candidate paths according to both distance and travel times (DI+TT), both distance and fuel consumption (DI+FC), both travel times and fuel consumption (TT+FC), and distance, travel times and fuel consumption (DI+TT+FC). When considering more than one travel cost, we consider each travel cost equally.

1) Distances (DI);
2) Travel times (TT);
3) Fuel consumption (FC);
4) Distance and travel times (DI+TT);
5) Distance and fuel consumption (DI+FC);
6) Travel times and fuel consumption (TT+FC);
7) Distance, travel times and fuel consumption (DI+TT+FC).

Baseline Regression Methods: To justify the effectiveness of *PathRank*, we consider six regression baselines.

1) Linear Regression (LR) [51];
2) Lasso Regression [52];
3) Support Vector Regression (SVR) [53];
4) Decision Tree Regression (DT) [54];
5) Decision Tree Regression with Adaboost (DTA) [55];
6) Long Short-Term Memory (LSTM) [56], we replace the bi-directional GRU units by LSTM units.

6.1.8 Implementation Details

All algorithms are implemented in Tensorflow. Code is available at https://github.com/Sean-Bin-Yang/Learning-to-Rank-Paths. We conduct experiments on a computer node on the CLAAUDIA cloud (www.claaudia.aau.dk), running Ubuntu 16.04.6 LTS, with one Intel(R) Xeon(R) CPU @2.50GHz and one Tesla V100 GPU card.

6.2 Verifying the Design Choices of *PathRank*

6.2.1 Effects of Training Data Generation Strategies

We investigate how the different training data generation strategies affect the accuracy of *PathRank*. We first consider *PR-A1*, where we only use graph embedding method node2vec to initialize the vertex embedding matrix B and do not update B during training.

Table 2 shows the results, where we categorize the training data generation strategies into three categories based on top-k paths, diversified top-k paths, and multi-cost, diversified top-k paths. For each category, the best results are highlighted with underline. The best results over all categories is also highlighted with bold. We also show results when the embedding feature sizes are $M = 64$ and $M = 128$, respectively.

The results show that (1) when using the diversified top-k paths for training, we have higher accuracy (i.e., lower MAE and MARE and larger τ and ρ) compared to when using top-k paths; (2) using multi-cost, diversified top-k paths achieves better accuracy compared to single-cost, diversified top-k paths, thus achieving the best results; (3) a larger embedding feature size M achieves better results.

TABLE 2: Training Data Generation Strategies, *PR-A1*

Strategies	M	MAE	MARE	τ	ρ
TkDI	64	0.1433	0.2300	0.6638	0.7044
	128	0.1168	0.1875	0.6913	0.7330
TkTT	64	0.1302	0.2090	0.6642	0.7046
	128	0.1181	0.1896	0.6818	0.7208
TkFC	64	0.1208	0.1940	0.6692	0.7131
	128	0.1257	0.2019	0.6699	0.7110
D-TkDI	64	0.1140	0.1830	0.6959	0.7346
	128	0.0955	0.1533	0.7077	0.7492
D-TkTT	64	0.1050	0.1686	0.7124	0.7554
	128	0.0974	0.1564	0.7271	0.7714
D-TkFC	64	0.1045	0.1678	0.7100	0.7544
	128	0.0900	0.1445	0.7238	0.7685
D-TkM	64	0.1077	0.1729	0.7261	0.7679
	128	**0.0792**	**0.1271**	**0.7478**	**0.7876**

Next, we consider *PR-A2*, where the graph embedding matrix B is also updated during training to fit better the ranking score regression problem. Table 3 shows the results. The three observations from Table 2 also hold for Table 3. In addition, *PR-A2*

achieves better accuracy than does *PR-A1*, meaning that updating embedding matrix B is useful.

TABLE 3: Training Data Generation Strategies, *PR-A2*

Strategies	M	MAE	MARE	τ	ρ
TkDI	64	0.1163	0.1868	0.6835	0.7256
	128	0.1130	0.1814	0.7082	0.7481
TkTT	64	0.1218	0.1956	0.6858	0.7282
	128	0.1161	0.1864	0.7026	0.7446
TkFC	64	0.1216	0.1952	0.6911	0.7321
	128	0.1082	0.1737	0.7070	0.7477
D-TkDI	64	0.0940	0.1509	0.7144	0.7532
	128	0.0855	0.1373	0.7339	0.7731
D-TkTT	64	0.1010	0.1622	0.7283	0.7693
	128	0.0997	0.1600	0.7169	0.7596
D-TkFC	64	0.0938	0.1506	0.7318	0.7743
	128	0.0809	0.1299	0.7386	0.7811
D-TkM	64	0.0966	0.1551	0.7393	0.7771
	128	**0.0725**	**0.1164**	**0.7528**	**0.7905**

From the above experiments, the multi-cost, diversified top-k strategy *D-TkM* is the most promising strategy. Thus, we only use *D-TkM* for the remaining experiments.

Next, we investigate the effects on the similarity threshold δ used in the diversified top-k path finding. Specifically, we consider two threshold values 0.6 and 0.8 and the results are shown in Table 4. When a smaller threshold is used, i.e., higher diversity in the top-k paths, the accuracy is improved.

TABLE 4: Effects of Similarity Threshold δ

	δ	M	MAE	MARE	τ	ρ
PR-A1	0.6	64	0.1006	0.1615	0.7321	0.7733
		128	0.0770	0.1237	0.7496	0.7874
	0.8	64	0.1077	0.1729	0.7261	0.7679
		128	0.0792	0.1271	0.7478	0.7876
PR-A2	0.6	64	0.0817	0.1311	0.7404	0.7792
		128	**0.0710**	**0.1140**	**0.7751**	**0.8109**
	0.8	64	0.0966	0.1551	0.7393	0.7771
		128	0.0725	0.1164	0.7528	0.7905

6.2.2 Effects of Vertex Embedding

We investigate the effects of different vertex embedding strategies. We consider *PR-B* where we just use a randomly initialized embedding matrix B, which totally ignores graph topology. For *PR-A1* and *PR-A2* where we both use node2vec to embed vertices. Here, we use node2vec to embed both weighted and unweighted graphs, respectively. When embedding weighted graphs, we simply use distances as edge weights.

Based on the results in Table 5, we observe the following. First, *PR-B* gives the worst accuracy: the estimated ranking scores have the largest errors in terms of both MAE and MARE; and the ranking based on estimated ranking scores deviates the most from the ground truth ranking in terms of both τ and ρ. This suggests that ignoring graph topology when embedding vertices is not a good choice.

Second, when embedding vertices using node2vec, whether or not considering edge weights does not significantly change the accuracy. Thus, it is not a significant design choice.

Third, *PR-A2* achieves the best accuracy in terms of both errors on estimated ranking scores and consistency between two rankings. Thus, this suggests that considering graph topology improves accuracy and updating the embedding matrix B according to the loss function on ranking scores makes the embedding matrix fit better the ranking score regression problem. This also suggests that, by including spatial properties in the loss function, it has a potential to tune the embedding matrix B to capture spatial properties, which in turn should improve ranking score regression. This is verified in the following experiments on the multi-task framework.

TABLE 5: Effects of Vertex Embedding Strategies

	Embedding	MAE	MARE	τ	ρ
PR-B	—	0.1159	0.1816	0.7233	0.7611
PR-A1	unweighted	0.0878	0.1410	0.7453	0.7852
	weighted	0.0792	0.1271	0.7478	0.7876
PR-A2	unweighted	0.0734	0.1178	**0.7640**	**0.8012**
	weighted	**0.0725**	**0.1164**	0.7528	0.7905

6.2.3 Effects of Multi-task Learning

In the following set of experiments, we study the effects of the proposed multi-task learning framework. In particular, we investigate how much we are able to improve when incorporating different spatial properties in the loss function to let the vertex embedding also consider spatial properties, which may potentially contribute to better ranking score regression.

We start by *PR-A2-M1*, which considers only one auxiliary task on reconstructing distances. This means that *PathRank* not only estimate the ranking score of a competitive path but also tries to reconstruct the distance of the competitive paths. Table 6 shows the results with varying α values. When $\alpha = 0$, the auxiliary task is ignored, which makes *PR-A2-M1* into *PR-A2*, i.e., its corresponding model with only the main task on estimating ranking scores. When $\alpha > 0$, i.e., the auxiliary task on distances is considered while learning, we observe that the estimated ranking scores are improved. In particular, the setting with $\alpha = 0.6$ gives the best results in terms both τ and ρ, indicating that the ranking w.r.t. the estimated ranking scores is more consistent with the ground truth ranking. When $\alpha = 0.8$, it achieves the smallest MAE and MARE. Both settings suggest that considering the additional auxiliary task on reconstructing distance helps improve the final ranking.

PR-A2-M2 includes two auxiliary tasks on reconstructing both distances and travel times, and *PR-A2-M3* includes three auxiliary tasks on reconstructing distances, travel times, and fuel consumption. All the three multi-task models show that considering spatial properties improve the final ranking. In particular, when considering all the three spatial properties give the best final ranking in terms of τ and ρ, i.e., achieving the most consistent ranking w.r.t. the ground truth ranking.

6.2.4 Effects of Contexts Embedding

We also investigate how much we improve when adding the contextual information to the basic framework. To this end, we consider the advanced framework *PRC* where we include the departure time feature $F(t)$ and driver feature $F(k)$. Table 7 shows that contextual information contributes to improve the overall accuracy. This also suggests that the proposed temporal graph embedding is effective. However, recall that the contextual information is an optional input. In case that departure time and driver identifiers are not provided as inputs, we can only use the basic framework.

TABLE 6: Effects of α, *PR-A2-Mx*

	α	MAE	MARE	τ	ρ
PR-A2	0	0.0725	0.1164	0.7528	0.7905
PR-A2-M1	0.2	0.0756	0.1214	0.7713	0.8057
	0.4	0.0704	0.1129	0.7765	0.8110
	0.6	0.0693	0.1113	0.7783	0.8141
	0.8	0.0680	**0.1029**	0.7712	0.8057
PR-A2-M2	0.2	**0.0653**	0.1048	0.7727	0.8089
	0.4	0.0701	0.1125	0.7869	0.8235
	0.6	0.0777	0.1247	0.7752	0.8100
	0.8	0.0807	0.1296	0.7616	0.7973
PR-A2-M3	0.2	0.0724	0.1162	0.7732	0.8092
	0.4	0.0740	0.1188	0.7711	0.8090
	0.6	0.0662	0.1063	**0.7923**	**0.8261**
	0.8	0.0695	0.1116	0.7842	0.8177

TABLE 7: Effects on Context Embeddings

	MAE	MARE	τ	ρ
PR-A2-M3	0.0662	0.1063	0.7923	0.8261
PRC	**0.0611**	**0.0929**	**0.8178**	**0.8454**

6.3 Comparison with Baselines

6.3.1 Comparison with Baseline Ranking Heuristics

We consider the baseline ranking heuristics covered in Section 6.1.7. For each heuristics, we obtain a ranking. Then, we compare the ranking with the ground truth ranking to compute the corresponding τ and ρ.

Table 8 shows the comparison, where we categorize the testing cases based on the distances of the lengths of their corresponding trajectory paths into three categories (0, 5], (5, 10], and (10, 15] km. The results show that the ranking obtained by the proposed framework *PRC* is clearly the best in all categories, suggesting that the proposed multitask framework outperforms baseline heuristics.

TABLE 8: Comparison with Baseline Ranking Heuristics

	(0, 5]	(5, 10]	(10, 15]
	τ/ρ	τ/ρ	τ/ρ
DI	0.7515/0.7806	0.6630/0.6860	0.3784/0.3370
TT	0.6776/0.7054	0.6712/0.7024	0.6053/0.6625
FC	0.6885/0.7192	0.3920/0.3986	0.0279/-0.0210
DI+TT	0.7146/0.7430	0.6681/0.6942	0.4919/0.4998
DI+FC	0.7200/0.7499	0.5275/0.5423	0.2032/0.1580
TT+FC	0.6831/0.7123	0.5316/0.5505	0.3166/0.3208
DI+TT+FC	0.7059/0.7351	0.5754/0.5957	0.3372/0.3262
PRC	**0.8239/0.8521**	**0.8115/0.8382**	**0.6497/0.6620**

6.3.2 Comparison with Regression Baselines

For the regression baseline methods covered in Section 6.1.7, we consider two different types of features.

- Basic features (BF): each path is represented as a 3-dimensional vector that represents its distance, travel time, and fuel consumption.
- Advanced features (AF): each path is represented as an $N \times M$ matrix, where N is the cardinality of the path. For each vertex in the path, we obtain a M-dimensional vector using node2vec. This makes an $N \times M$ matrix.

Table 9 shows the comparison. The results show that the ranking obtained by the proposed framework *PRC* outperforms all baselines. This suggests that simply using basic features and advanced features do not offer meaningful representations for ranking paths. Our design on path representation that captures both road network topology and spatial properties is effective.

TABLE 9: Comparison with Regression Baselines

	Method	MAE	MARE	τ	ρ
BF	**LR**	0.2640	0.4012	0.6879	0.7150
	Lasso	0.2876	0.4371	0.6245	0.6678
	SVR	0.2390	0.3632	0.6543	0.6683
	DT	0.2516	0.3824	0.6530	0.6777
	DTA	0.2686	0.4082	0.6784	0.7135
AF	**LR**	0.3430	0.5213	0.0864	0.0854
	Lasso	0.2955	0.4484	0.6260	0.6686
	SVR	0.3369	0.5120	0.0857	0.0846
	DT	0.4141	0.6284	0.0450	0.0693
	DTA	0.4301	0.6527	0.0812	0.0395
Deep Learning	**LSTM**	0.2682	0.4076	0.4569	0.4619
	PRC	**0.0611**	**0.0929**	**0.8178**	**0.8454**

6.4 Comparison with Driver Specific *PathRank*

We investigate if driver specific models are able to provide more accurate personalized ranking. We select two drivers with the largest amount training trajectories. Driver 1 has 2,068 trajectories and Driver 2 has 1,457 trajectories. We train two driver-specific *PRC* models, *PRC-Dr1* and *PRC-Dr2*, using only the trajectories from the corresponding driver. In addition, we consider a baseline *BA* from a personalized routing algorithm [21], which learns a 3-dimensional vector to combine the distance, travel time, and fuel consumption of a path to derive a personalized cost for the path. Then, the paths are ranked according to their personalized costs. The vector is learned from individual drivers' trajectories and thus different drivers have different vectors. *BR-Dr1* and *BR-Dr2* use trajectories from the corresponding drivers to learn the vectors.

We test the models on two testing sets $Dr1$ and $Dr2$ which consist of testing trajectories from Driver 1 and Driver 2, respectively. Table 10 shows that for the testing trajectories from Driver 1, *PRC-Dr1* outperforms *PRC-Dr2*; and for the testing trajectories from Driver 2, *PRC-Dr2* outperforms *PRC-Dr1*. This is not surprising and this indicates that different drivers do have different preferences; and a ranking model trained on one driver may not provide accurate ranking for a different driver. In addition, *PRC-Dr1* outperforms *BA-Dr1* and *PRC-Dr2* outperforms *BA-Dr2*, indicating that the proposed *PathRank* outperforms the baseline. Note that *BA* ranks path according to the personalized costs but does not estimate the ranking scores. Thus, *BA* has no MAE and MARE values but only τ and ρ values.

The proposed *PRC* performs the best on both testing sets. This suggests that the proposed method is able to learn a better ranking when using much more trajectories from different drivers. Together with the context embedding, it enables accurate personalized ranking.

Next, we report statistics on a case-by-case comparison, where Table 11 shows the percentages of the cases where a driver specific *PathRank* outperforms *PRC*. Specifically, *PRC-Dr1* outperforms *PRC* in ca. 22% of the testing cases from Driver 1, and *PRC-Dr2* outperforms *PRC* in ca. 19% of the testing cases from Driver 2.

The results from the above two tables suggest that user-specific *PathRank* models still have a potential to achieve personalized ranking, which may outperform the *PathRank* model trained on all trajectories, i.e., *PRC*. We plan to explore attention mechanisms on driver feature vectors to achieve this in future work.

TABLE 10: Comparison with Driver Specific *PathRank*

Testing Data	Model	MAE	MARE	τ	ρ
Dr1	*PRC-Dr1*	0.1037	0.1532	0.8395	0.8531
	PRC-Dr2	0.2557	0.3975	0.6544	0.6419
	PR-A2-M3	0.0786	0.1162	0.8309	0.8513
	PRC	**0.0658**	**0.0972**	**0.8466**	**0.8710**
	BA-Dr1	—	—	0.7298	0.7392
Dr2	*PRC-Dr1*	0.1476	0.2182	0.7741	0.7851
	PRC-Dr2	0.1079	0.1677	0.8535	0.8750
	PR-A2-M3	0.0851	0.1323	0.8571	0.8900
	PRC	**0.0625**	**0.0971**	**0.8668**	**0.8945**
	BA-Dr2	—	—	0.7573	0.7800

TABLE 11: Percentage when *PR-Dr* Outperforms *PRC*

PR-Dr1		*PR-Dr2*	
τ	ρ	τ	ρ
22.70%	22.70%	19.31%	18.81%

6.5 Effects on Training Data Size

We conduct the next experiment to investigate the performance when varying the sizes of training data. Specifically, we use 25%, 50%, 75%, 100% of the total training data to train *PRC*, respectively. Based on the results shown in Table 12, more training data gives better performance.

TABLE 12: Effects of the Size of Training Data

Percentage	MAE	MARE	τ	ρ
25%	0.1071	0.1672	0.7574	0.7898
50%	0.0871	0.1323	0.7873	0.8179
75%	0.0892	0.1355	0.7928	0.8227
100%	**0.0611**	**0.0929**	**0.8178**	**0.8454**

6.6 Online Efficiency

Since ranking candidate paths is conducted online, we report the runtime. Table 13 reports the runtime for estimating a path when using different *PathRank* models. It shows that the non-multi-task learning models, i.e., *PR-B*, *PR-A1*, and *PR-A2*, have similar run time. Multi-task learning models take longer time and the more auxiliary tasks are included in a model, the longer time the model takes. *PRC* takes the longest time, on average 58.2 ms. Suppose that an advanced routing algorithm or a commercial navigation system returns 10 candidate paths, *PRC* is able to return a ranking in 58.2 ms on average, which is within a reasonable response time.

TABLE 13: Average Testing Runtime Per Path (ms)

PR-B	*PR-A1*	*PR-A2*	*PR-A2-M1*	*PR-A2-M2*	*PR-A2-M3*	*PRC*
11.4	11.3	11.5	22.8	34.4	45.1	58.2

6.7 Offline Training Efficiency

We study training efficiency by varying road network graph sizes and path lengths. First, we consider three road network graphs with different sizes in Table 14. When the road network graph has 1 million vertices, each epoch takes 24.7 seconds, which is still within a reasonable time. Note that the main application scenario is intra-city, where multiple path candidates connecting the same source and destination exist. When traveling inter-cities, there are often very few path alternatives, e.g., using highways. Thus, a road network graph with 1 million vertices should already be able to model a very large city.

TABLE 14: Effects of Graph Size for Training Time

# of Vertices	*50K*	*500K*	*1000K*
Run time per epoch (s)	8.3	12.1	24.7

Next, Table 15 shows the training time when the training paths are with different numbers of vertices. For long paths with more vertices, the RNN needs to go through more GRU units. Thus, it takes longer time.

TABLE 15: Effects of Path Lengths, Training Time

# of vertices per path	*60*	*120*	*180*	*240*
Run time per epoch (s)	7.8	12.6	21.3	24.8

7 Conclusion and Future Work

We propose a context-aware, multitask learning framework to rank paths in road networks. We propose an effective method to generate a compact and diverse set of training paths to enable efficient and effective learning. Then, we propose a multi-task learning framework to enable road network embedding that takes into account spatial properties. A recurrent neural network, together with the learned road network embedding, is employed to estimate the ranking scores which eventually enable ranking paths. In addition, a temporal graph is proposed to embed temporal contexts. Empirical studies conducted on a large real world trajectory set demonstrate that the proposed framework is effective and efficient for practical usage. As future work, it is of interest to exploit different means, e.g., attention mechanisms on path lengths and outlier trajectories removal [57], [58], to further improve the ranking quality of *PathRank*. It is also of interest to explore parallel computing [59] to improve efficiency.

References

[1] C. Guo, C. S. Jensen, and B. Yang, "Towards total traffic awareness," *SIGMOD Record*, vol. 43, no. 3, pp. 18–23, 2014.

[2] C. Guo, B. Yang, J. Hu, and C. S. Jensen, "Learning to route with sparse trajectory sets," in *ICDE*, 2018, pp. 1073–1084.

[3] J. Hu, C. Guo, B. Yang, and C. S. Jensen, "Stochastic weight completion for road networks using graph convolutional networks," in *ICDE*, 2019, pp. 1274–1285.

[4] Z. Ding, B. Yang, Y. Chi, and L. Guo, "Enabling smart transportation systems: A parallel spatio-temporal database approach," *IEEE Trans. Computers*, vol. 65, no. 5, pp. 1377–1391, 2016.

[5] V. Ceikute and C. S. Jensen, "Routing service quality - local driver behavior versus routing services," in *MDM*, 2013, pp. 97–106.

[6] B. Yang, C. Guo, C. S. Jensen, M. Kaul, and S. Shang, "Stochastic skyline route planning under time-varying uncertainty," in *ICDE*, 2014, pp. 136–147.

[7] J. Y. Yen, "Finding the k shortest loopless paths in a network," *management Science*, vol. 17, no. 11, pp. 712–716, 1971.

[8] H. Liu, C. Jin, B. Yang, and A. Zhou, "Finding top-k shortest paths with diversity," *IEEE Trans. Knowl. Data Eng.*, vol. 30, no. 3, pp. 488–502, 2018.

[9] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *KDD*, 2016, pp. 855–864.

[10] S. B. Yang and B. Yang, "Learning to rank paths in spatial networks," in *ICDE*, 2020, pp. 2006–2009.

[11] Y. Lei, W. Li, Z. Lu, and M. Zhao, "Alternating pointwise-pairwise learning for personalized item ranking," in *CIKM*, 2017, pp. 2155–2158.

[12] Z. Hu, Y. Wang, Q. Peng, and H. Li, "Unbiased lambdamart: An unbiased pairwise learning-to-rank algorithm," in *WWW*, 2019, pp. 2830–2836.

[13] Q. Ai, K. Bi, J. Guo, and W. B. Croft, "Learning a deep listwise context model for ranking refinement," in *SIGIR*, 2018, pp. 135–144.

[14] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: online learning of social representations," in *KDD*, 2014, pp. 701–710.

[15] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, "Line: Large-scale information network embedding," in *WWW*, 2015, pp. 1067–1077.

[16] S. Cao, W. Lu, and Q. Xu, "Deep neural networks for learning graph representations," in *AAAI*, 2016, pp. 1145–1152.

[17] H. Wang, J. Wang, J. Wang, M. Zhao, W. Zhang, F. Zhang, X. Xie, and M. Guo, "Graphgan: Graph representation learning with generative adversarial nets," *CoRR*, vol. abs/1711.08267, 2017.

[18] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," in *ICLR*, 2013.

[19] J. Hu, B. Yang, C. Guo, and C. S. Jensen, "Risk-aware path selection with time-varying, uncertain travel costs: a time series approach," *VLDB J.*, vol. 27, no. 2, pp. 179–200, 2018.

[20] C. Guo, B. Yang, J. Hu, C. S. Jensen, and L. Chen, "Context-aware, preference-based vehicle routing." *VLDB Journal*, online first, 2020.

[21] B. Yang, C. Guo, Y. Ma, and C. S. Jensen, "Toward personalized, context-aware routing," *VLDB J.*, vol. 24, no. 2, pp. 297–318, 2015.

[22] S. A. Pedersen, B. Yang, and C. S. Jensen, "Fast stochastic routing under time-varying uncertainty." *VLDB Journal*, vol. 29, no. 4, pp. 819–839, 2020.

[23] J. Hu, B. Yang, C. S. Jensen, and Y. Ma, "Enabling time-dependent uncertain eco-weights for road networks," *GeoInformatica*, vol. 21, no. 1, pp. 57–88, 2017.

[24] S. A. Pedersen, B. Yang, and C. S. Jensen, "Anytime stochastic routing with hybrid learning," *PVLDB*, vol. 13, no. 9, pp. 1555–1567.

[25] B. Yang, C. Guo, and C. S. Jensen, "Travel cost inference from sparse, spatio-temporally correlated time series using markov models," *PVLDB*, vol. 6, no. 9, pp. 769–780, 2013.

[26] B. Yang, J. Dai, C. Guo, C. S. Jensen, and J. Hu, "PACE: a path-centric paradigm for stochastic path finding," *VLDB J.*, vol. 27, no. 2, pp. 153–178, 2018.

[27] S. A. Pedersen, B. Yang, and C. S. Jensen, "A hybrid learning approach to stochastic routing," in *ICDE*, 2020, pp. 2010–2013.

[28] T. Anwar, C. Liu, H. L. Vu, and M. S. Islam, "Roadrank: Traffic diffusion and influence estimation in dynamic urban road networks," in *CIKM*, 2015, pp. 1671–1674.

[29] T. Kieu, B. Yang, C. Guo, and C. S. Jensen, "Distinguishing trajectories from different drivers using incompletely labeled trajectories," in *CIKM*, 2018, pp. 863–872.

[30] J. Won, S. Kim, J. Baek, and J. Lee, "Trajectory clustering in road network environment," in *CIDM*, 2009, pp. 299–305.

[31] Y. Li, R. Yu, C. Shahabi, and Y. Liu, "Diffusion convolutional recurrent neural network: Data-driven traffic forecasting," in *ICLR*, 2018.

[32] Z. Pan, Y. Liang, W. Wang, Y. Yu, Y. Zheng, and J. Zhang, "Urban traffic prediction from spatio-temporal data using deep meta learning," in *KDD*, 2019, pp. 1720–1730.

[33] J. Hu, C. Guo, B. Yang, C. S. Jensen, and H. Xiong, "Stochastic origin-destination matrix forecasting using dual-stage graph convolutional, recurrent neural networks," in *ICDE*, 2020, pp. 1417–1428.

[34] R.-G. Cirstea, B. Yang, and C. Guo, "Graph attention recurrent neural networks for correlated time series forecasting." in *MileTS19@KDD*, 2019.

[35] R. Cirstea, D. Micu, G. Muresan, C. Guo, and B. Yang, "Correlated time series forecasting using multi-task deep neural networks," in *CIKM*, 2018, pp. 1527–1530.

[36] M. Hua and J. Pei, "Probabilistic path queries in road networks: traffic uncertainty aware path selection," in *EDBT*, 2010, pp. 347–358.

[37] K. Mouratidis, Y. Lin, and M. L. Yiu, "Preference queries in large multi-cost transportation networks," in *ICDE*, 2010, pp. 533–544.

[38] H. Liu, C. Jin, B. Yang, and A. Zhou, "Finding top-k optimal sequenced routes," in *ICDE*, 2018, pp. 569–580.

[39] G. Andonov and B. Yang, "Stochastic shortest path finding in path-centric uncertain road networks," in *MDM*, 2018, pp. 40–45.

[40] P. Newson and J. Krumm, "Hidden markov map matching through noise and sparseness," in *SIGSPATIAL*, 2009, pp. 336–343.

[41] E. Erkut and V. Verter, "Modeling of transport risk for hazardous materials," *Operations research*, vol. 46, no. 5, pp. 625–642, 1998.

[42] T. Joachims, "Optimizing search engines using clickthrough data," in *KDD*, 2002, pp. 133–142.

[43] Z. Cao, T. Qin, T. Liu, M. Tsai, and H. Li, "Learning to rank: from pairwise approach to listwise approach," in *ICML*, 2007, pp. 129–136.

[44] J. Hershberger, M. Maxel, and S. Suri, "Finding the k shortest simple paths: A new algorithm and its implementation," *ACM Trans. Algorithms*, vol. 3, no. 4, p. 45, 2007.

[45] J. Dai, B. Yang, C. Guo, and Z. Ding, "Personalized route recommendation using big trajectory data," in *ICDE*, 2015, pp. 543–554.

[46] D. Erhan, Y. Bengio, A. C. Courville, P. Manzagol, P. Vincent, and S. Bengio, "Why does unsupervised pre-training help deep learning?" *J. Mach. Learn. Res.*, vol. 11, pp. 625–660, 2010.

[47] K. Cho, B. van Merrienboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," in *EMNLP*, 2014, pp. 103–111.

[48] Y. Li, K. Fu, Z. Wang, C. Shahabi, J. Ye, and Y. Liu, "Multi-task representation learning for travel time estimation," in *KDD*, 2018, pp. 1695–1704.

[49] C. Guo, Y. Ma, B. Yang, C. S. Jensen, and M. Kaul, "Ecomark: evaluating models of vehicular environmental impact," in *SIGSPATIAL*, 2012, pp. 269–278.

[50] C. Guo, B. Yang, O. Andersen, C. S. Jensen, and K. Torp, "Ecomark 2.0: empowering eco-routing with vehicular environmental models and actual vehicle fuel consumption data," *GeoInformatica*, vol. 19, no. 3, pp. 567–599, 2015.

[51] Y. Shi, J. Li, and Z. Li, "Gradient boosting with piece-wise linear regression trees," *arXiv preprint arXiv:1802.05640*, 2018.

[52] S. Wang, B. Ji, J. Zhao, W. Liu, and T. Xu, "Predicting ship fuel consumption based on lasso regression," *Transportation Research Part D: Transport and Environment*, vol. 65, pp. 817–824, 2018.

[53] T. Chen and S. Lu, "Accurate and efficient traffic sign detection using discriminative adaboost and support vector regression," *IEEE Trans. Vehicular Technology*, vol. 65, no. 6, pp. 4006–4015, 2016.

[54] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T. Liu, "Lightgbm: A highly efficient gradient boosting decision tree," in *NIPS*, 2017, pp. 3146–3154.

[55] S. Y. Kim and A. Upneja, "Predicting restaurant financial distress using decision tree and adaboosted decision tree models," *Economic Modelling*, vol. 36, pp. 354–362, 2014.

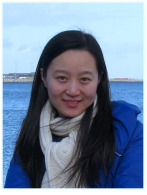
[56] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[57] T. Kieu, B. Yang, and C. S. Jensen, "Outlier detection for multidimensional time series using deep neural networks," in *MDM*, 2018, pp. 125–134.

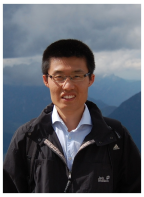
[58] T. Kieu, B. Yang, C. Guo, and C. S. Jensen, "Outlier detection for time series with recurrent autoencoder ensembles," in *IJCAI*, 2019, pp. 2725–2732.

[59] P. Yuan, C. Sha, X. Wang, B. Yang, A. Zhou, and S. Yang, "XML structural similarity search using mapreduce," in *WAIM*, 2010, pp. 169–181.

Sean Bin Yang received his bachelor and master degrees from Chongqing University of Posts and Telecommunications in 2012 and 2015, respectively. He is a Ph.D. student in the Department of Computer Science, Aalborg University, Denmark. His research interests include machine learning and spatio-temporal data mining.



Chenjuan Guo is an Associate Professor at Aalborg University, Denmark. She received her Ph.D. degree in computer science from the University of Manchester, UK, in 2011. Her research interests include spatial-temporal data management, heterogeneous data management, and data analytics.



Bin Yang is a Professor at Aalborg University, Denmark. He was previously at Aarhus University, Denmark and at MaxPlanck-Institut fur Informatik, Germany. He received his Ph.D. degree in computer science from Fudan University, China. His research interests include machine learning and data management.